Practical: News ML

Davin Jeong, Charles Zhou
djeong@college.harvard.edu, czhou@college.harvard.edu

1 Part A: Feature Engineering, Baseline Models

1.1 Approach

We implemented two approaches for feature representation of text: CountVectorizer and TfidfVectorizer. Both generate vectors the size of the text vocabulary. CountVectorizer weighs words by their frequency, while TfidfVectorizer weighs words by ”importance,” using a formula which assigns words appearing frequently in one document and in-frequently in others a high importance. Both are computationally-efficient but low-level methods. CountVectorizer is useful in applications where the frequency of uncommon words says a lot about the classes, for instance in spam detection, but fails to utilize context/word order. TfidfVectorizer focuses on the importance between words making it slightly more useful for learning the relevance of words, but again does not extract contextual or semantic information.

We used the following parameters when implementing the tokenizers on scikit-learn:

- CountVectorizer: **ngram_range**: [1, 1]), **max_features**: 10,000
- TfidfVectorizer: **ngram_range**: [1, 2], **max_features**: 57,500, **token_pattern**: r”\(?u\)\b\w+\b—\[ˆ\w\s\]”, **sublinear_tf** = True

`ngram_range` describes the upper and lower-limit of n-grams that can be extracted. `token_pattern` describes what a ”token” may represent. `max_features` restricts the size of the feature space. `sublinear_tf` applies scaling to reduce the dominance of very frequent terms in Tfidf-based methods. All of these parameters were chosen at the start and then fixed after experimentation on how they affected noise and over-fitting.

Details on the multi-class logistic regression model are provided below.

For this section, we implemented these methods using scikit-learn with default parameters. As suggested, we did not perform any hyperparameter tuning on our models.

1.2 Results

Vectorizer	**Overall**	ALJAZ	BBCNEWS	BLOOMBERG	CNBC	CNNW	CSPAN	CSPAN2
Count	**52.32**	17.24	65.68	82.94	69.92	55.71	36.76	39.39
Tfidf	**64.07**	13.79	96.68	93.53	80.47	89.04	56.37	58.79

	CSPAN3	DW	FBC	FOXNEWSW	GBN	KDTV	KGO	KNTV	KPIX	KQED	KRON
Count	21.47	41.30	69.16	44.80	92.86	87.50	43.97	35.68	33.90	11.11	49.01
Tfidf	2.82	17.39	78.97	61.20	85.71	90.62	36.17	55.68	28.81	5.56	66.89

Table 1: Table of the validation accuracy in % using baseline methods

1.3 Discussion

In general, the performance of our models was not fantastic, with performances certainly below what would be needed to reliably trust their outputs. We hypothesize that the baselines performed as poorly as they did because of differences in training and validation distributions. When running our models, we found that a lot of labels in training did not show up often if at all in validation and vice versa; as a result, these differences likely had both CountVectorizer and TfidfVectorizer trained

models struggling to fit the channels. Conversely, we hypothesize that our models performed well on channels like BLOOMBERG and BBCNEWS because of consistent word composition. We found that these focus on politics and usually mention words like "trump," "biden," "policies," while other channels focus on technology like "ai," "cyber," etc. They likely helped our classifier easily differentiate those channels.

TfidfVectorizer performed better than CountVectorizer overall (52.32% vs 64.07%) and across the majority of classes (11/18). Intuitvely, this made sense to us because TfidfVectorizer takes into account, not only word frequency, but also importance; so our models would've had more signal to work with. However, Tfidf did not outperform Count across all channels; and we hypothesize that this discrepancy may come from variances in word composition. The number of words used only in Train was 47923 and the number of words used only in Validation was 4581. As a result, TfidfVectorizer may have had trouble assigning importance, since some channels may focus on certain words but these channels may have only one or two samples. Distribution shift may have resulted in incorrect understandings of word importance based on the training set; which wouldn't have troubled CountVectorizer but would have hindered Tfidf.

2 Part B: More Modeling

2.1 First Step

2.1.1 Approach

We decided to use Random Forests, which are ensenble/non-linear models often used for their resistance againnst over-fitting and interpretable structure. More details on the model are provided in the appendix.

2.1.2 Results

We trained RFs for both Count and Tfidf, but did not optimize hyperparameters.

Vectorizer	**Overall**	ALJAZ	BBCNEWS	BLOOMBERG	CNBC	CNNW	CSPAN	CSPAN2
Count	**35.90**	0.00	76.75	72.35	35.16	88.58	7.35	0.61
Tfidf	**41.49**	0.00	98.15	87.06	41.80	95.89	17.16	0.61

Vectorizer	CSPAN3	DW	FBC	FOXNEWSW	GBN	KDTV	KGO	KNTV	KPIX	KQED	KRON
Count	2.26	0.00	52.34	8.80	28.57	81.25	24.82	16.22	1.69	0.00	33.77
Tfidf	2.26	0.00	59.35	10.40	28.57	93.75	16.31	23.78	0.00	0.00	27.15

2.1.3 Discussion

Both of the RF models performed worse overall than the logistic regression baselines from Part A. Specifically, while logistic regression achieved around 52–64% validation accuracy, the Random Forests achieved only 36–41% accuracy depending on the vectorizer. This could be because RFs struggle due to high dimensionality and sparse feature space of the data. In our code, we had 10000 max features for CountVectorizer and 57500 max features for Tfidf, which are sparse and noisy. Since RFs try to split data based on single-feature comparisons, it may have been difficult to find robust splits without dimensionality reduction (e.g. if we conducted PCA). In contrast, logistic regression is more robust to sparse high-dim data and finds simpler, linear decision boundaries.

Additionally, the class imbalance and distribution shift between training and validation likely exacerbated RF's weaknesses since trees trained on underrepresented classes likely do not generalize well. Overall, these results suggest that non-linear models like Random Forests require careful tuning or preprocessing to outperform strong linear baselines in text classification tasks.

2.2 Hyperparameter Tuning and Validation

2.2.1 Approach

We conducted hyperparameter tuning for multiple models. We present two below: one as a natural extension of previous parts, and the other due to its high performance (others are included in our attached notebook: Random Forests (RF) and Support Vector Machines (SVM). Based on B1 results, we continued using CountVectorizer for Random Forest, as it had shown relatively better performance, and switched to TfidfVectorizer for SVM (from examining a few test trials).

For Random Forest, we used RandomizedSearchCV, which randomly sampled and compared 10 configurations from the following parameter space:

$$\begin{aligned} \texttt{n_estimators} &: [50, 100, 200, 300, 1000] \\ \texttt{max_depth} &: [10, 20, 30, \text{None}] \\ \texttt{min_samples_split} &: [2, 5, 10] \end{aligned}$$

For SVM, we tuned a LinearSVC classifier on Tfidf features, as specified before. Our search space included:

$$\begin{aligned} \texttt{C} &: [0.01, 0.1, 1, 10, 100] \\ \texttt{loss} &: [\text{hinge}, \text{squared_hinge}] \\ \texttt{penalty} &: [\text{l2}] \\ \texttt{dual} &: [\text{True}, \text{False}] \\ \texttt{max_iter} &: [1000, 5000, 10000, 100000] \end{aligned}$$

We did not include larger C since we ran individual test trials, finding that training took a long time while delivering worse performances. After identifying promising regions, we refined our search further, testing around $C = 1$ manually. We ultimately selected $C = 0.99$ with `class_weight='balanced'` and `max_iter=100000`, which achieved the highest validation accuracy.

2.2.2 Results

Below are our results for our best RF and SVM models (validation accuracy in %):

Model	**Overall**	1TV	ALJAZ	BBCNEWS	BLOOMBERG	CNBC	CNNW	COM	CSPAN	CSPAN2	CSPAN3	DW	FBC	FOXNEWSW
RF	**56.37**	0.00	17.24	100.00	92.35	66.02	88.58	0.00	35.78	40.61	2.26	10.87	66.36	47.60
SVM	**69.17**	0.00	58.62	95.57	93.53	84.77	70.78	0.00	57.35	63.64	6.78	60.87	85.51	68.80

Model	GBN	KDTV	KGO	KNTV	KPIX	KQED	KRON	KSTS	KTVU	MSNBCW	PRESSTV	RT	RUSSIA24	SFGTV
RF	64.29	96.88	27.66	52.43	27.12	5.56	47.68	90.91	62.61	65.92	0.00	0.00	0.00	8.11
SVM	100.00	93.75	46.10	61.08	54.24	22.22	84.11	95.45	73.91	73.54	0.00	17.07	0.00	43.24

2.2.3 Discussion

We expected the tuned models to outperform both the baselines and the untuned models because they are more expressive and benefited from optimization. The RF model improved significantly from 41.49% to 56.37% validation accuracy. Our best model was our tuned SVM which showed substantial improvements, achieving a validation accuracy of 69.17%, compared to the greatest previous accuracy of 64.207% from logistic regression. This demonstrates that regularization combined with well-tuned hyperparameters can yield significant gains in classification performance.

For SVMs, tuning the regularization parameter C was very impactful. A slightly smaller C (like 0.99) allowed the model to generalize better to validation data without overfitting to noisy

patterns in the training set. The use of `class_weight='balanced'` also helped reduce the effects of class imbalance, which seemed to improve recall for underrepresented channels. The rest of the parameters worked best with their default (`penalty=l2, loss=squared_hinge, dual=false`).

For Random Forest, the best parameters were `n_estimators=300, min_samples_split=2, max_depth=None`. Increasing the number of trees and allowing deeper splits led to better performance across frequent classes like BLOOMBERG, CNNW, and FBC. However, the model still struggled with rare or ambiguous classes, which is likely due to insufficient signal in high-dimensional (sparse feature spaces where many features are irrelevant or noisy). Without dimensionality reduction, Random Forests are vulnerable to overfitting on large class signals.

Our validation strategy relied on validation accuracy as our primary metric. Although we used RandomizedSearchCV to explore the hyperparameter space, we also manually refined promising regions to avoid wasting resources on suboptimal ranges. Ultimately, the results show the importance of both tuning and model choice. While the models in general all benefitted from tuning, we found linear SVMs to be the best approach we tried for the high-dimensional, sparse text data in this task.

3 Final Write-up and Reflections

3.1 Discussion:

Here, we provide some explanation and reflection on our steps through the modelling workflow.

1. Data Pipeline: In our exploration, we employed both bag of words based representations, which we ultimately selecting, and LLM-based embeddings, which are perhaps more timely and relevant in the current day. However, since our transcriptions were noisy and imperfect, the LLMs struggled to distinguish semantic signal from noise; so the first approach turned out to be more relevant towards our application.

2. Model Selection: The main trade-off that we had to work around was expressivity and overfitting. Many non-linear models like MLPs captured the training set distribution but weren't able to generalize on the validation set, as a result of the class/distribution shift. We found that linear models, especially SVMs, managed this tradeoff the best: managing to capture the relationship well while having a natural regularizing effect due to its simplicity.

3. Model Tuning: Since the written task was to optimize for held-out accuracy, we optimized our model on our validation set, testing various configurations and comparing their validation accuracies. We tested out validation, cross-validation, and ensembling. Ensembling often resulted in underfitting and didn't perform very well; however, cross-validation and validation strategies, on top of our RandomSearch on a large parameter space, helped tremendously at increasing our validation accuracy.

4. Bias-Variance Trade-off: As mentioned above, we were very cognizant of this tradeoff, as many tested models overfit the data as a result of the distribution shifts. To find the optimal tradeoff, we selected for simpler model classes and used hyperparameter search on the validation set, to get a sense of how well our model maintained this tradeoff.

5. Evaluation: As the task was to optimize for test accuracy, we evaluated our models' performance on validation accuracy and per-class accuracy. The advantage was that we got a sense

of how well we were doing with respect to our target; but the drawback was that our metrics didn't account for the entire behavior of the model (for instance, if some classes were actually more important to classify). We also considered evaluating precision/recall, but we decided to stick with the target goal of accuracy.

6. Domain-specific Evaluation: We took a look at the domain distribution and some samples before beginning our modeling, to inform our exploration. One key choice was about including punctuation in our feature space–to make sure that we accounted for grammatical structure in our bigrams. Another key observation was that our input data was very noisy and had many transcription errors; so we wanted to find a robust, simpler model class. Though the overall problem framing for the task of predicting channels made sense, we found that our architecture and learned weights didn't provide a lot of intuitive clarity on decision making. Most notably, our model seemed heavily influenced by how datasets reflected different time-periods, which meant that decision making wouldn't be generalizable for other times. Regardless, one way of interpreting our resulting weights might be to evaluate which bigrams have the most weight, and to see if they make sense with respect to the task.

7. Design Review: One issue with our model choice and data was distribution shift: the presence of channels, the confounders like ongoing news cycles, and the natural flow of events meant that our model might perform well on one slice, but might not be generalizable for future periods. With more time, we would have combined semantic and non-semantic features, so that our model could learn more about the underlying language, which may be a more robust feature than the presence of specific bigrams.

8. Execution & Implementation: Supposing that we wanted to build and deploy this model in the real-world, we would have to consider how big our model can be, which is severely limited by training resources, and how much data we can collect, for instance whether the data collection procedure described in the handout is scalable. However, there are also ethical considerations to be made about whether or not this model should actually be deployed, especially without the permission of the channels themselves. For instance, if we were working for a channel, and wanted a method to systematically identify and leave bad ratings/hate speech on specific channels, this would be an immoral deployment of the technology; so we perhaps shouldn't produce such a model.

4 Optional Exploration, Part C: Explore some more!

4.1 Approach

We explored a variety of modeling strategies and preprocessing methods beyond the main pipeline to test the limits of performance and gain insight into what techniques perform well.

- **Distilled BERT Embeddings:** We used the `distilbert-base-uncased` model from the HuggingFace `transformers` library to extract 768-dimensional contextual embeddings for each clip. For each text snippet, we took the token representation (from the final hidden layer) as a fixed-length summary embedding. We then used these embeddings as our features and trained models on them (e.g. logistic regression, RF, multi-layer perceptron). For logistic regression, we tested over the parameters `solver='saga', penalty='l1, l2', C=0.25`.

For our MLP, we passed these embeddings to a model with three hidden layers (528, 256, 128), ReLU activations, and L2 regularization, trained using cross-entropy loss and Adam optimizer. While this approach gave slightly better per-class calibration, it struggled to outperform simpler models.

- **XGBoost:** We also experimented with `XGBoostClassifier` using Tfidf features. We tuned over the number of estimators, learning rate, and max depth using a small grid. XGBoost handled some minority classes better than Random Forest, but it still underperformed compared to our best SVM model. We hypothesize this is due to overfitting in high-dimensional sparse settings without tailored feature selection.

- **Ensembling:** We tried soft-voting ensembling of multiple classifiers (Logistic Regression, MLP, and RF), averaging predicted class probabilities (both naive and weighted averaging). While this improved validation accuracy on a few edge cases, the overall gain was negligible compared to the single best model (sometimes performing worse).

- **Class Reweighting and Sampling:** To reduce the effects of class imbalance (which we found to be prevalent in the training data), we tested `class_weight='balanced'` in both SVMs and Logistic Regression, and tested upsampling low-frequency classes in the training set. Class weighting generally improved validation accuracy on rare classes (e.g., KQED, PRESSTV) without harming precision on common classes.

- **Custom Vocabulary and Feature Pruning:** We also experimented with more aggressive vocabulary control in CountVectorizer and TfidfVectorizer by limiting 'max_features' to between 10,000 and 25,000 and enabling 'stop_words='english''. We tried collecting the top non-english words, turning it into a vocabulary, and re-extracting features. This had negligible and sometimes negative effects on model accuracy.

- **Neural Networks:** In addition to using BERT embeddings, we trained standard MLP classifiers on basic Tfidf features. We did a hyperparameter search for both logistic regression and MLPs, and found little success. Overall, the best performing architecture we found consisted of two hidden layers (256, 128) with ReLU activations. While this model had more flexibility than others, its performance was highly sensitive to regularization and initialization. It did not outperform SVMs but had similar performance to logistic regression.

- **Naive Bayes:** We trained a Multinomial Naive Bayes classifier on both Count and basic Tfidf features. It trained extremely fast but performed poorly on minority classes due to its strong conditional independence assumption. Overall, it performed the worst among the models we trained, though we did not do a hyperparameter search.

- **K-Nearest Neighbors (KNN):** We tried KNN classifiers using cosine similarity on basic Tfidf vectors. The models performed very poorly, similar to Naive Bayes. Even with small values of k (e.g., 5), the model failed to generalize, suggesting that KNN is not good in sparse, high-dimensional environments.

- **Feature Stacking:** We attempted feature concatenation by h-stacking basic and engineered Tfidf vectors with count vectors. The resulting hybrid vectors had tens of thousands of dimensions, potentially resulting in overfitting. Despite the added information, models trained

on stacked features typically performed much worse than those using single-source representations like engineered Tfidf alone.

- **Dimensionality Reduction:** We noticed that the basic vectorizers extracted a lot of features so we explored:
  - **Variance Thresholding:** We filtered out features with very low variance across the training set. This marginally helped with little visible difference in validation accuracy.
  - **Truncated SVD:** We applied truncated singular value decomposition (SVD) on Tfidf features, projecting them to 100–5000 components (different tests). This improved runtime, but seemed to lose structure needed classification, so the models we tested on this performed worse.

An example of one of these attempts can be found in the Appendix, where we include the loss and accuracy curves for training MLP on distilled bert embeddings. Overall, none of the results from this extra exploration outperformed the fine-tuned SVM in terms of validation accuracy. While more complex models and stacked features introduced capacity, they were often less reliable. Well-tuned linear models with regularization, feature engineering, and class balancing remained the most reliable and highest validation accuracy for this task.

The model that we submitted for the Kaggle competition was our SVM trained on the Tfidf features we engineering at the beginning with the hyperparameters in part B. This proved to have the highest 5-fold cross-validation data

Appendices

A Math For Models

A.1 Multi-Class Classification Logistic Regression

We used a simple multi-class logistic regression model to predict output probabilities for each class. According to scikit-learn's documentation, our classifier predicts class probabilities of the following form, where W is a matrix of coefficients for each dimension/class,

$$P(y_i = k|X_i) = \frac{\exp(X_i W_k + W_{0,k})}{\sum_{k=0}^{K-1} \exp(X_i W_l + W_{0,l})}$$

On a high-level, our model learns hyperplanes as the boundary between two classes and predicts output probabilities using a softmax on logits of the linear model. The objective of the optimization is

$$\min_W -\frac{1}{S} \sum_{i=1}^{n} \sum_{k=0}^{K-1} s_{ik} [y_i = k] log(P(y_i = k|X_i)) + \frac{r(W)}{SC}$$

or a multi-class cross entropy loss maximizing the probability you assign to the true class. Here, s are the weights of classes assigned by the user, $r(W)$ is the choice of regularization term, and C is the regularization coefficient; these allow us to account for problems like over-fitting and class-imbalances.

A.2 Random Forests

RFs are ensemble models that combine the predictions of multiple decision trees to make a final prediction where each tree is trained on a random subset of the data (uses random subset of features for splits as well). Typically, each split in the tree aims to have around a 50-50 partition of the remaining data space, and at any individual "layer" in the tree, the choice of path is based off of a single metric. The output is at the leaf of whichever path is taken. Thus, an RF is much better than individual trees since it reduces variance through ensembling:

$$\hat{y} = majority_vote(h_1(x), h_2(x), \ldots, h_N(x))$$

where $h_n(x)$ is the nth decision tree for N total trees (which in our case we set). We initialized the RF with 100 estimators and a maximum depth of 20, choosing these values based on intuitive defaults: a moderately deep tree can capture some non-linear structure without severely overfitting, and an ensemble size of 100 is likely enough for a stable majority vote. We did not perform any hyperparameter tuning.

A.3 Support Vector Machines

We also trained a linear Support Vector Machine (SVM) using the `LinearSVC` classifier from scikit-learn. Unlike logistic regression, which models class probabilities via softmax, the linear SVM aims to find the hyperplane that maximizes the margin between classes.

In the binary case, the optimization objective is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where ξ_i are slack variables that allow for misclassification, and C is the regularization strength controlling the trade-off between margin size and classification error.

In the multi-class setting, `LinearSVC` uses a one-vs-rest scheme, where a separate binary classifier is trained for each class k, treating $y_i = k$ as positive and $y_i \neq k$ as negative. At prediction time, the class with the highest decision function score is selected:

$$\hat{y}_i = \arg\max_k \; \mathbf{w}_k^\top \mathbf{x}_i + b_k$$

Although `LinearSVC` does not output probabilities, it can be more robust in high-dimensional sparse settings due to its margin-based objective. More specifically, it not only attempts to classify data, but it attempts to find the best separation of classes, which helps produce generalizable decision boundaries in the high-dimensinoal scenario. The model benefits from regularization and class weighting to handle overfitting and imbalanced data.

B Figures

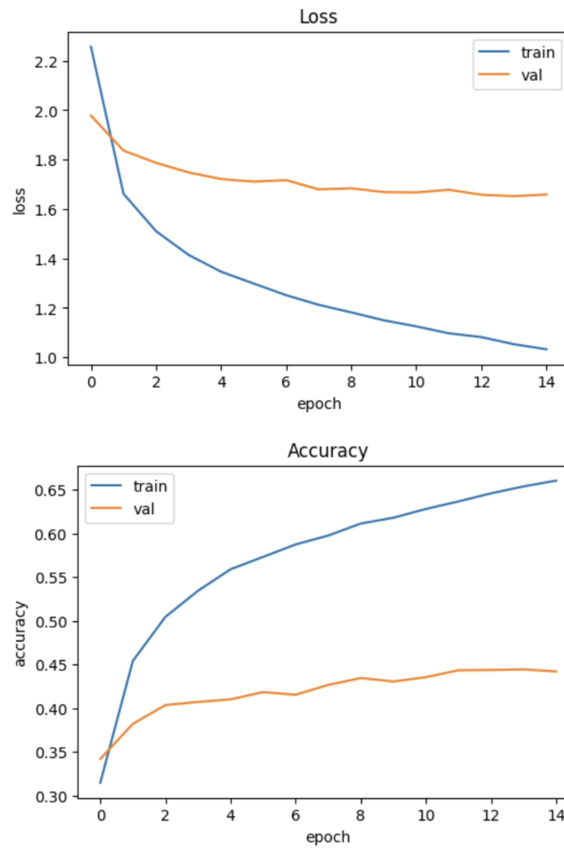

Figure 1: Training an MLP on Distilled Roberta